第十三节 Spring 基础架构重构

主要使用场景

服务端视图渲染

模板引擎

- JSP
  - 基于 Servlet 引擎 - JspServlet
- Thymeleaf
- Freemarker
- Velocity
  - Maven 手动创建工程

Mybatis-Generator

模板设计模式

- 模板
  - 表达式 - 逻辑、循环、赋值等指令
  - 占位符（变量）- EL 表达式
    - 变量名关联变量
    - 变量利用 JavaBeans
  - 上下文 - Key-Value 的存储结构
    - Spring MVC Model 类似于 Map
  - 执行引擎
    - Velocity - TemplateEngine
- 国际化 - i18n
  - accept-language: en,zh;q=0.9,zh-TW;q=0.8,zh-CN;q=0.7
  - JDK- ResourceBundle
  - Spring - MessageSource
    - ResourceBundle
    - MessageFormat
- 多主题切换
  - 门户网站

REST 服务端

MVC 模式 -> MC， 对 V 不敏感

在早期 Spring Web MVC -> JsonView

HTTP 方法

- GET
- POST

URI

- URI Template
  - 常亮路径 - /users/list
  - 变量路径 - /users/{id}
    - Spring MVC - @PathVariable
    - JAX-RS - @PathParam

@MatrixParam

内容协商

客户端请求服务端以某种媒体类型来响应

对于服务端而言，响应内容文本和二进制

REST application/json 或者 application/xml

客户端而言，请求头 Accept

服务端而言，响应头 Content-Type

MediaType(主类型/子类型)

Spring MVC

JAX-RS

序列化和反序列化

反序列化 -> HTTP 请求 -> 文本或者二进制 -> POJO

序列化 -> HTTP 响应 -> POJO -> 文本或者二进制

> 类型的概念 - 类型转换器
>
> byte[] -> POJO
>
> POJO -> byte[]

核心组件

Frontend Controller - DispatcherServlet

initStrategies 方法

子流程套路

- 通过 Spring IoC 容器进行类型的依赖查找或者名称+类型依赖查找
- （可选）通过指定Bean 名称 + 类型依赖查找
- 通过 defaultStrategies 做兜底方案

处理方法映射：HandlerMapping

@RequestMapping 与 HandlerMethod 做一个映射

Handler == HandlerMethod = 应用标注 @RequestMapping 或者派生注解 Java 方法

```
@Controller
public class AbcController {

    @RequestMapping(value="/abc")
    public String abc(@RequestParam String
name){
        return "abc";
```

```
    }

    @RequestMapping(value="/abc1")
    public View abc1(@RequestParam String name)
{
        return new View("abc");
    }

     @RequestMapping(value="/abc2")
    public void abc2(@RequestParam String
name,ModelAndView modelAndView){
        modelAndView.setView("abc")
    }
}
```

Spring Web MVC 基于 Servlet 引擎，参数 HttpServletRequest 和 HttpServletResponse

以 JSP 为例，Spring Web MVC 的跳转等价于 DispatcherServelt forward 到 JspServlet

@Component

@Controller

@Service

@Repository

HTTP 请求 -> URI -> @RequestMapping -> 执行具体方法

方法参数处理?

适配：HandlerAdapter

HandlerMethod 与 Servlet API 做适配，利用 HandlerMethod 中的执行结果，去控制适配 ServletRequest 和 ServletResponse。

Handler 内建实现有两种：

- org.springframework.web.servlet.mvc.Controller
- 标注 @RequestMapping 方法

HandlerAdapter 存在多种实现，比如最常用注解实现：

- org.springframework.web.servlet.mvc.Controller
  - org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter（1.0+）
- @RequestMapping
  - org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter（2.5 - 3.1）
  - org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter （3.1+）

HandlerAdapter 提高优先级：

- 内部实现通常增加 setOrder(int) 方法，来提供 getOrder() 放的数据
- 自定义组件通过实现 Ordered 接口或者标注 @Order
- （不安全方式）通过定义 Bean 的顺序

执行：HandlerExecutionChain

猜想是责任链模式，类似于 Filter 与 Servlet 关系，FilterChain

潜在风险

```
    @Nullable
    public HandlerInterceptor[]
getInterceptors() {
        if (this.interceptors == null &&
this.interceptorList != null) {
            this.interceptors =
this.interceptorList.toArray(new
HandlerInterceptor[0]);
        }
        return this.interceptors;
    }
```

1. interceptors 是一个能被修改得数组，比如：getInterceptors()[0] = null;

2. HandlerInterceptor 尽量不要有状态，比如：getInterceptors()[0].setValue(...)

处理方法 HandlerMethod 拦截器 - HandlerInterceptor

- 前置判断 - preHandle
  - 当且仅当方法返回 true 时，执行 HandlerMethod
- 后置处理 - postHandle
  - HandlerMethod 已经被执行，其执行结果为 ModelAndView
    - 当 ModelAndView 参数为空，说明是非视图渲染，即 REST 场景（@since 2.5）
    - 否则，就是视图渲染
- 完成回调 - afterCompletion
  - 正常 preHandle -> handle -> postHandle -> afterCompletion
  - preHandle 失败 -> afterCompletion

```
public class MyFilter implements Filter {


    public void doFilter(ServletRequest
request,ServletResponse response){
        try{
            if(preHandle()){
                handler();
```

```
                postHandle();
            }
        }finally{
            // afterCompletion
        }
    }
}
```

模型和视图组合类 - ModelAndView

组成成员：

- 视图对象 - view 字段（Object）：一种可能是 View 对象，一种是 String 对象
- 模型对象 - Model 对象
    - org.springframework.ui.ModelMap 类 - LinkedHashMap
    - org.springframework.ui.Model 接口

请求属性 - org.springframework.web.context.request.RequestAttributes

org.springframework.web.context.request.ServletRequestAttributes

Bean Scope：

- Singleton（默认）
- Prototype（原型）
- request（Bean 存放在 HttpServletRequest）
- session（Bean 存放在 HttpSession）
- application（Bean 存放在 ServletContext）
  - Bean 存放在ServletContext 主要是给 Servlet 视图
    - Spring IoC 关联的 Servlet - DispatcherServlet
    - Jsp 的 Servlet - JspServlet
  - 一个 Servlet 应用对应了一个 ServletContext，但是它可能对应多个 Spring BeanFactory（ApplicationContext）
    - ContextLoaderListener（Root ApplicationContext）
    - DispatcherServlet（Child ApplicationContext）

自定义组件： WebMvcConfigurer

Spring 5 之前通常使用 WebMvcConfigurerAdapter 类

Spring 5 开始直接使用 WebMvcConfigurer 接口

Spring Framework 留给应用程序扩展 Web MVC 特性的，WebMvcConfigurer Bean 不是必须

WebMvcConfigurer 引导类 - DelegatingWebMvcConfiguration

DelegatingWebMvcConfiguration （@Configuration Class）会被 @EnableWebMvc 引导

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Documented
@Import(DelegatingWebMvcConfiguration.class)
public @interface EnableWebMvc {
}
```

> @Configuration 标注的类会被 CGLib 提升
>
> @Import 将一个普通的类，注册为 Spring Bean，类似于 Configuration Class
>
> - Lite（轻量级） - 没有标注 @Configuration，它不会被 CGLib，它不会生成代理类
> - Full（全量及）- Configuration Class 会被 CGLib 代理，生成代理类

DelegatingWebMvcConfiguration 继承了 WebMvcConfigurationSupport，其中 WebMvcConfigurationSupport 定义了

- HandlerAdapter Bean - RequestMappingHandlerAdapter

```
    @Bean
    public RequestMappingHandlerAdapter
requestMappingHandlerAdapter() {
        ...
    }
```

- HandlerMapping Bean
  - RequestMappingHandlerMapping

```
    @Bean
    public RequestMappingHandlerMapping
requestMappingHandlerMapping() {
        ...
    }
```

- BeanNameUrlHandlerMapping

Servlet 引擎静态资源 Servlet 处理器 - org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler

HandlerMethodArgumentResolver

提供方法参数获取元信息，并且将 Servlet API 中的信息作为方法参数填充

```
@RestController
public class AbcController {

    @RequestMapping(value="/abc/{def}")
    public String abc(@RequestParam String
name,@PathVariable String def){
        return "abc";
    }
}
```

HandlerMethod 方法上参数注解如何处理

@RequestParam - ServletRequest#getParameter(String)

- 利用 Java 反射 API Method
  - 获取参数注解 - Annotation[][] getParameterAnnotations()
    - 方法允许多个参数，一个参数允许标注不同的注解
  - 获取参数名称 - Parameter[] getParameters()

    - 获取方法参数名称
- 实现 - org.springframework.web.method.annotation.RequestParamMethodArgumentResolver

@PathVariable - @RequestMapping URI Template - /abc/{def}

- 参数类型转换 - HttpMessageConverter

HandlerMethodReturnValueHandler

HttpMessageConverter

HTTP Servlet 相关的数据源类型转化成目标参数或者 HandlerMethod 返回值类型

Servlet 相关数据类型

HttpServletRequest#getParameter -> java.lang.String

HttpServletRequest#getHeader -> java.lang.String

HttpServletRequest#getIntHeader-> int

HttpServletRequest#getDateHeader-> long

HttpServletRequest#getInputStream() -> java.io.InputStream

Spring Framework 内建了常见的数据类型装换，HttpMessageConverter 实现

String -> 简单类型

String -> POJO（Jackson 来实现）

Spring Web MVC 默认实现 - org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport#getMessageConverters

```
    protected final List<HttpMessageConverter<?>> getMessageConverters() {
        if (this.messageConverters == null) {
            this.messageConverters = new ArrayList<HttpMessageConverter<?>>();
            // 应用配置的 HttpMessageConverter
            configureMessageConverters(this.messageConverters);
            if (this.messageConverters.isEmpty()) {
                // Spring Web MVC 内建 HttpMessageConverter 实例
                addDefaultHttpMessageConverters(this.messageConverters);
            }
            // 应用扩展的 HttpMessageConverter
```

```
extendMessageConverters(this.messageConverters)
;
            }
            return this.messageConverters;
        }
```

相关技术

- 类型转换器
- 序列化和反序列化

异常处理器 - HandlerExceptionResolver

校验器 - Validator

Spring Validator 适配 Bean Validation - org.springframework.validation.beanvalidation.LocalValidatorFactoryBean

Servlet 与 JSP 基础

Jsp 四大范围

页面范围：仅限于当前 JSP 页面

PageContext#setAttribute(String,Object)

请求范围：当前请求（可能跨多个 Servlet ）

ServletRequest 上下文

ServletRequest#setAttribute(String,Object)

会话范围：用户 HttpSession

Session上下文

HttpSession#setAttribute(String,Object)

应用范围：ServletContext

应用上下文

ServletContext#setAttribute(String,Object)

####

JSP 九大对象